Some New Attacks upon Security Protocols

Gavin Lowe
Oxford University Computing Laboratory, Wolfson Building,
Parks Road, Oxford, OX1 3QD, United Kingdom.
gavin.lowe@comlab.ox.ac.uk

October 1, 1996

Abstract

Many security protocols have appeared in the literature, with aims such as agreeing upon a cryptographic key, or achieving authentication. However, many of these have been shown to be flawed. In this paper we present a number of new attacks upon security protocols, and discuss ways in which we may avoid designing incorrect protocols in the future.

1 Introduction

Many security protocols have appeared in the literature; these have various aims, such as agreeing upon a cryptographic key, or achieving authentication, where each agent becomes assured of the other's identity. Unfortunately, a large proportion of these protocols are subject to attacks, leading to them not correctly achieving their goals. In this paper, we present a few more attacks upon such protocols.

The main point of this paper is to highlight the fact that, despite much research on the subject, many insecure protocols are still being produced. Further, most of the weaknesses that allow the attacks are well known. Our hope is that by highlighting these errors—once again—protocol designers will avoid making the same mistakes in future. We will have more to say on this subject in the concluding section.

Our notation is very standard. The protocols are between agents A and B, sometimes with the help of a trusted server S. We denote nonces by N_a, etc.; the subscripts denote their origin. We denote a key shared between A and B by K_{ab}, etc. We denote message m encrypted with key k by $\{m\}_k$. We denote message m signed by agent A by $S_A(m)$: we do not specify how this signing is implemented. We denote an attacker by I; when the attack impersonates an agent A to send a message, we denote this I_A; we use the same notation to denote I intercepting a message intended for A. When an attack requires several runs of the protocol, we will denote the runs α, β, etc., and denote the messages $\alpha.1$, etc.

Some of the attacks we present are implementation-dependent, and we will describe necessary conditions for the attacks to succeed. We believe that a protocol specification can be considered secure only if all its sensible implementations are secure: if the security of a protocol depends upon certain assumptions about the way it is implemented, then those assumptions need to be clearly stated.

Further, we believe that it is a bad idea to base the security of a protocol upon an assumption that is very hard to implement. For example, the original design for the Kerberos protocol [21, 29] stated that live authenticators should be stored to detect replay attacks. However, Bellovin and Merritt [3] report that this was not implemented in early versions of the protocol, and point out that a security feature is not very useful if it is too hard to implement. We believe that it is better to design a protocol in such a way that its security does not depend upon such implementation requirements.

In the next section we define precisely what we mean by the term *authentication*. In the following sections, we present attacks upon four different protocols. We close by summing up, and suggesting ways of producing more reliable protocols.

2 What is authentication?

Most of the errors caused by the attacks in this paper are errors of authentication, and so we make clear here what we understand by the term "authentication".

Our definition of authentication is as follows:

> Whenever an agent A completes a run if the protocol, apparently with B, then B has recently been running the protocol, apparently with A, and the two agents agree upon who initiated the run, and agree upon all data values used in the run; further, there is a one-one relationship between the runs of A and the runs of B.

Our definition of authentication is quite strong, but we believe that it is the right one. It essentially insists that, on completion, the two agents agree on each other's state. (In fact, the question of a one-one relationship between runs will play no part in this paper; we include it above only for completeness.)

This definition of authentication is weaker than Roscoe's *intensional* specification [25], which essentially states that the pattern of messages must be as the protocol designer expected: in other words, every message sent by A must be received by B, and vice versa, and the send- and receive-events must be interleaved in the expected order. Similarly, it is weaker than Gollmann's specification G4 in [12]:

> The origin of all messages in the protocol has to be authenticated.

In fact, most of the attacks we present will break a weaker definition of authentication:

> Whenever an agent A completes a run if the protocol, apparently with B, then B has recently been running the protocol, apparently with A.

This specification allows cases where B agrees that it has been running the protocol with A, but the two agents disagree about some of the details; for example: both agents may believe that it was they who initiated the protocol run; or the agents may disagree over the value of some nonce exchanged during the run. In these cases, it would be incorrect for either agent to take some action in the resulting session that depended upon those details.

In a couple of cases, the attacks break an even weaker definition of authentication, which we consider to be the weakest reasonable definition:

> Whenever an agent A completes a run if the protocol, apparently with B, then B has recently been running the protocol.

This weaker specification allows the case where B thinks he has been running the protocol with some other agent, and may never have heard of A. It is the same as Gollmann's specification G3 [12].

3 The Woo and Lam mutual authentication protocol

The following protocol was presented in [32]. It aims to establish a session key and provide mutual authentication between two agents A and B, with the help of a trusted server S.

$$
\begin{array}{lrcl}
\text{Message 1.} & A \rightarrow B & : & A, N_a \\
\text{Message 2.} & B \rightarrow A & : & B, N_b \\
\text{Message 3.} & A \rightarrow B & : & \{A, B, N_a, N_b\}_{K_{as}} \\
\text{Message 4.} & B \rightarrow S & : & \{A, B, N_a, N_b\}_{K_{as}}, \{A, B, N_a, N_b\}_{K_{bs}} \\
\text{Message 5.} & S \rightarrow B & : & \{B, N_a, N_b, K_{ab}\}_{K_{as}}, \\
& & & \quad \{A, N_a, N_b, K_{ab}\}_{K_{bs}} \\
\text{Message 6.} & B \rightarrow A & : & \{B, N_a, N_b, K_{ab}\}_{K_{as}}, \{N_a, N_b\}_{K_{ab}} \\
\text{Message 7.} & A \rightarrow B & : & \{N_b\}_{K_{ab}}.
\end{array}
$$

If A wants to establish a session with B, he sends him a nonce (message 1). B responds by returning another nonce (message 2). A then prepares an encrypted message for the server, containing the two agents' identifiers and the two nonces, and sends it to B (message 3). B forwards this message to the server, along with a similar message of his own creation (message 4). When the server receives this message, he checks that the agent identities and nonces agree, and if so selects a new key K_{ab} to be used in the subsequent session between A and B. The server creates two encrypted components, one for each of A and B, and returns both to B (message 5). B decrypts his component to obtain the key, and then forwards the other component to A, along with a component encrypted with K_{ab} to convince A that he knows the key (message 6). Finally, A decrypts his component to learn the key, and returns a message to B to convince him that he knows the key (message 7).

The attack we will present depends upon a couple of assumptions. We assume that nonces and keys are simply sequences of bits, without any typing information; hence any such bit sequence of the right length will be accepted as a nonce or a key. Further, we assume that the bit sequences representing agent identifiers, nonces and keys are all of the same length, which means, for example, that a nonce may be accepted as a key. We believe that making such assumptions is very reasonable, because it seems perfectly valid to implement the protocol in this way: nothing in the protocol description prohibits such an implementation. If the protocol designer believes that certain implementation considerations are necessary, then these should be made explicit.

In the following, "bit-string", "bit-string′", etc. represent any sequences of bits that may be accepted as being of the form $\{A, B, N_a, N_b\}_k$ by an agent who does not possess the key k—that is, any bit sequences of the appropriate length; in the attack, no agent ever tries to decrypt these messages, so any such sequences of bits will do. The attack can be described as follows:

$$
\begin{array}{lrcl}
\text{Message } \alpha.1. & I_A \to B & : & A, B \\
\text{Message } \alpha.2. & B \to I_A & : & B, N_b \\
\text{Message } \alpha.3. & I_A \to B & : & \text{bit-string} \\
\text{Message } \alpha.4. & B \to I_S & : & \text{bit-string}, \{A, B, B, N_b\}_{K_{bs}} \\
\text{Message } \beta.1. & I_A \to B & : & A, N_b \\
\text{Message } \beta.2. & B \to I_A & : & B, N_b' \\
\text{Message } \beta.3. & I_A \to B & : & \text{bit-string}' \\
\text{Message } \beta.4. & B \to I_S & : & \text{bit-string}', \{A, B, N_b, N_b'\}_{K_{bs}} \\
\text{Message } \alpha.5. & I_S \to B & : & \text{bit-string}'', \{A, B, N_b, N_b'\}_{K_{bs}} \\
\text{Message } \alpha.6. & B \to I_A & : & \text{bit-string}'', \{B, N_b\}_{N_b'} \\
\text{Message } \alpha.7. & I_A \to B & : & \{N_b\}_{N_b'}.
\end{array}
$$

In message $\alpha.1$, the intruder starts to impersonate A to attack B, and chooses to send B's own identity as a nonce. B responds in message $\alpha.2$ by returning a nonce N_b. The intruder then sends in message $\alpha.3$ any bit-string of the appropriate length. B creates an encrypted component of the appropriate form, and tries forwarding this to the server in message $\alpha.4$, but the intruder intercepts this message.

The intruder cannot immediately create an appropriate message 5, but instead starts a second run, β, again impersonating A, so as to use B as an oracle. He sends, in message $\beta.1$, the nonce N_b that B chose for run α. B returns a nonce (message $\beta.2$), and the intruder again responds with a bit-string of the appropriate length (message $\beta.3$). B creates an encrypted component containing the two agent identities and the two nonces, and tries sending it to the server, along with the previous message, but the intruder intercepts the message (message $\beta.4$). This message is of precisely the form expected by B in message $\alpha.5$—it contains the identity A, the "nonce" B, the nonce N_b, and a component, N_b', that may be interpreted as a key; hence the intruder forwards this message to B, and B accepts N_b' as a session key.

Finally, A sends an appropriate message (message $\alpha.6$), and the intruder returns the component $\{N_b\}_{N_b'}$ (message $\alpha.7$), which he can produce because he knows the "key" N_b', thus completing the handshake.

The reason this attack worked is that the intruder was able to replay the encrypted component from message $\beta.4$ and have it accepted as part of message $\alpha.5$. The weakness that allowed the attack is the fact that two different encrypted components have very similar forms—indeed, it was spotting this weakness that made us decide to look for an attack on the protocol. The reason this leads to an error of authentication is that these two components are intended to mean different things; the intruder is able to swap these components to have B interpret message $\alpha.5$ in a way in which it was not intended. It is worth noting that the BAN Logic "proof" of the protocol produces different idealizations of these two components. This fact highlights an assumption of BAN Logic that when an agent receives a message, he understands the context in which that message was sent; clearly, this assumption is not always valid. This fact also serves to remind us that the most error-prone part of BAN Logic is the idealization process.

To prevent the attack, we need to make sure that the different encrypted components in the protocol have distinct forms, for example by swapping the order of the fields, or by adding some redundancy.

The idea of having a nonce interpreted as a key is not new. It has occurred previously in an attack on the adapted Yahalom protocol [7] in [31], and attacks on the Neuman-Stubblebine protocol [23], reported in [30, 14].

4 The adapted Andrew protocol

In [26], the Andrew secure RPC protocol was introduced, which allows two agents, who already share a key K_{ab}, to agree upon a new session key K'_{ab}, and to perform an authentication handshake. In [7], this protocol was analysed using BAN Logic, and a weakness exposed. A correction to the protocol was suggested:

$$
\begin{array}{lll}
\text{Message 1.} & A \rightarrow B & : \ A, N_a \\
\text{Message 2.} & B \rightarrow A & : \ \{N_a, K'_{ab}\}_{K_{ab}} \\
\text{Message 3.} & A \rightarrow B & : \ \{N_a\}_{K'_{ab}} \\
\text{Message 4.} & B \rightarrow A & : \ N_b.
\end{array}
$$

If A wants to establish a new key with B, he sends a nonce N_a (message 1). B responds by choosing a new key K'_{ab}, and returning it to A, along with the nonce, encrypted with the existing key (message 2). A decrypts this message to learn the new key, and then sends a message back to B, encrypted with the new key, to demonstrate to B that he knows the new key (message 3). Finally, B returns a new nonce N_b to A, to tell A that he is free to proceed (message 4).

One peculiarity with this protocol is the role of message 4: it seems to carry no information. Indeed, there is a very simple attack where the intruder intercepts this message, and changes its value before forwarding it to A. It is arguable whether this should be considered an attack; however, it *is* an attack using our strong definition of authentication, and it is clear that

A should not perform any subsequent actions that rely upon B knowing the value of N_b.

Ignoring the above comments, there is another attack upon the protocol, using two parallel runs. In run α, agent A tries to contact B, but an intruder I intercepts the messages, and impersonates B. In run β, the intruder I impersonates B so as to initiate a session with A. The intruder uses A as an oracle in order to encrypt messages.

$$\begin{array}{lrcl}
\text{Message } \alpha.1. & A \rightarrow I_B & : & A, N_a \\
\text{Message } \beta.1. & I_B \rightarrow A & : & B, N_a \\
\text{Message } \beta.2. & A \rightarrow I_B & : & \{N_a, K'_{ab}\}_{K_{ab}} \\
\text{Message } \alpha.2. & I_B \rightarrow A & : & \{N_a, K'_{ab}\}_{K_{ab}} \\
\text{Message } \alpha.3. & A \rightarrow I_B & : & \{N_a\}_{K'_{ab}} \\
\text{Message } \beta.3. & I_B \rightarrow A & : & \{N_a\}_{K'_{ab}} \\
\text{Message } \alpha.4. & I_B \rightarrow A & : & N_i \\
\text{Message } \beta.4. & A \rightarrow I_B & : & N'_a.
\end{array}$$

When A seeks to start a session with B, sending nonce N_a, the intruder intercepts the message (message $\alpha.1$), and impersonates B to try to start a new session, sending the same nonce N_a (message $\beta.1$). A reacts to this by sending a new key K'_{ab}, and the nonce, both encrypted with K_{ab} (message $\beta.2$); the intruder then replays this as its next message in run α (message $\alpha.2$). In reply, A returns an encrypted copy of N_a (message $\alpha.3$), which the intruder again replays, as its next message in run β (message $\beta.3$). Finally both agents send new nonces as the final messages in each run (messages $\alpha.4$ and $\beta.4$). Hence, A believes that he has established a session with B in run α, and he believes that B has established a session with him in run β, even though B may in fact be absent.

Bird et al. [5, 6] have presented a number of similar attacks, where messages are simply reflected back at the sender. Similar attacks are possible against any protocol using symmetric key encryption where no encrypted component contains an indication of its intended direction.

The problem is easily fixed by adapting the protocol, for example by changing message 2 to include an encrypted copy of the sender's identity:

$$\text{Message 2.} \quad B \rightarrow A \ : \ \{N_a, K'_{ab}, B\}_{K_{ab}}.$$

This will prevent the above attack from working, because message $\beta.2$ will include the identity A, so the intruder will be unable to replay the message as message $\alpha.2$.

5 The station-to-station protocol

The *station-to-station protocol* [11] consists of Diffie-Hellman key establishment followed by mutual authentication. The key establishment assumes the existence of a publically-known cyclic group and a corresponding primitive element α.

$$\begin{array}{lrcl} \text{Message 1.} & A \to B & : & A, B, \alpha^x \\ \text{Message 2.} & B \to A & : & B, A, \alpha^y, \{S_B(\alpha^y, \alpha^x)\}_k \\ \text{Message 3.} & A \to B & : & A, B, \{S_A(\alpha^x, \alpha^y)\}_k. \end{array}$$

(The original description of the protocol did not include the "A" and "B" fields in the messages, but we have included them to make the subsequent explanations clearer.) A chooses a value x, and sends the value α^x to B (message 1). B then chooses a value y, and sends the value α^y to A. Both agents calculate the value of a key $k = \alpha^{xy} = (\alpha^x)^y = (\alpha^y)^x$. B returns to A a signed copy of the two exponentials, encrypted under the key k (message 2). A returns a similar message to B (message 3).

The attack on the protocol proceeds as follows:

$$\begin{array}{lrcl} \text{Message } \alpha.1. & A \to I_B & : & A, B, \alpha^x \\ \text{Message } \beta.1. & I \to B & : & I, B, \alpha^x \\ \text{Message } \beta.2. & B \to I & : & B, I, \alpha^y, \{S_B(\alpha^y, \alpha^x)\}_k \\ \text{Message } \alpha.2. & I_B \to A & : & B, A, \alpha^y, \{S_B(\alpha^y, \alpha^x)\}_k \\ \text{Message } \alpha.3. & A \to I_B & : & A, B, \{S_A(\alpha^x, \alpha^y)\}_k. \end{array}$$

A tries to start a run of the protocol with B, sending the exponential α^x, but I intercepts the message (message $\alpha.1$). The intruder then starts a second run, sending the same exponential α^x to B, using his own identity (message $\beta.1$). B responds with an exponential α^y, and the appropriate encrypted component (message $\beta.2$). The intruder simply forwards this message to A (message $\alpha.2$), who accepts that he is communicating with B. Finally, A signs the two exponentials, and tries returning them to B (message $\alpha.3$).

This is an attack using our definition of authentication, because A thought he was running the protocol with B, while B thought he was running the protocol with I, and may never have heard of A. The consequences are not serious, since the intruder does not learn the value of the key. However, this certainly represents an attack, since it leads to A holding incorrect beliefs.

In [11], Diffie et al. define authentication as follows: if A accepts the identity of B (that is, A successfully completes the run), then B's record of the run must *match* A's: every message that B sends or receives during the run must correspond to a message received or sent by A, and vice versa. This attack breaks that definition of authentication, since A's and B's records of the protocol runs clearly do not match.

It is worth noting that if the fields representing the agents' identities were removed from the protocol (as is the case in the original protocol description) then there it an essentially identical attack:

$$\begin{array}{lrcl}
\text{Message } \alpha.1. & A \rightarrow I_B & : & \alpha^x \\
\text{Message } \beta.1. & I \rightarrow B & : & \alpha^x \\
\text{Message } \beta.2. & B \rightarrow I & : & \alpha^y, \{S_B(\alpha^y, \alpha^x)\}_k \\
\text{Message } \alpha.2. & I_B \rightarrow A & : & \alpha^y, \{S_B(\alpha^y, \alpha^x)\}_k \\
\text{Message } \alpha.3. & A \rightarrow I_B & : & \{S_A(\alpha^x, \alpha^y)\}_k.
\end{array}$$

However, this new attack does *not* meet the definition in [11], because the authors do not consider the sender and recipient of a message to be part of an agent's record of the run; thus, when A accepts B's identity (after message $\alpha.2$), B's record of run β matches A's record of run α, even though B thought he was running the protocol with I rather than with A. I consider this to be an error: this must be considered an attack for the same reasons as above, since A ends up holding incorrect beliefs. To argue that the sender and recipient fields in protocol messages should not be considered part of the protocol is to miss the point: the main aim of an authentication protocol is to establish the identities of the agents involved; the protocol should be designed so that if an intruder tampers with these identities, then nobody should be fooled.

The attack is easily prevented, by including A's identity in the signed part of message 2; similarly, it is prudent to include B's identity in the signed part of message 3:

$$\begin{array}{lrcl}
\text{Message 1.} & A \rightarrow B & : & A, B, \alpha^x \\
\text{Message 2.} & B \rightarrow A & : & B, A, \alpha^y, \{S_B(A, \alpha^y, \alpha^x)\}_k \\
\text{Message 3.} & A \rightarrow B & : & A, B, \{S_A(B, \alpha^x, \alpha^y)\}_k.
\end{array}$$

6 The KSL protocol

In [15], Kehne, Schönwälder and Landendörfer, introduced two protocols for multiple authentications. The first protocol establishes a session key to be shared between two agents, establishes a ticket (which is used in subsequent re-authentications), and performs an authentication handshake. The second protocol uses the ticket from the first protocol to re-establish the key, and performs an authentication handshake.

The ticket is simply a message component, created by B and sent to A, which A can later use to re-establish the key. It is of the form $\{T_b, A, K_{ab}\}_{K_{bb}}$, where: T_b is a generalized timestamp, containing the timestamp from B's clock at the time at which the key was created, a lifetime, limiting the validity of the ticket, and a time identifier, which is used to prevent certain attacks based on B's clock being altered; A is the identity of the agent with whom B shares the key; K_{ab} is the shared key; and K_{bb} is a key known only to B.

The initial protocol to establish a shared key and a ticket is as follows:

$$\begin{array}{lll}
\text{Message 1.} & A \to B & : N_a, A \\
\text{Message 2.} & B \to S & : N_a, A, N_b, B \\
\text{Message 3.} & S \to B & : \{N_b, A, K_{ab}\}_{K_{bs}}, \{N_a, B, K_{ab}\}_{K_{as}} \\
\text{Message 4.} & B \to A & : \{N_a, B, K_{ab}\}_{K_{as}}, \{T_b, A, K_{ab}\}_{K_{bb}}, \\
 & & \quad N_b', \{N_a\}_{K_{ab}} \\
\text{Message 5.} & A \to B & : \{N_b'\}_{K_{ab}}.
\end{array}$$

A sends a nonce to B in message 1. B forwards this, along with his own nonce, to the server S. The server chooses a key K_{ab}, and prepares encrypted components for each of A and B: these each contain the key, the identity of the agent with whom the key is shared, and the appropriate nonce, encrypted with the key shared between the agent and the server. The server sends both these components to B in message 3. B decrypts his component to learn the key. In message 4, he forwards the component meant for A, along with a ticket, a new nonce (for a nonce challenge), and A's nonce encrypted with the key (to prove that he knows the key). Finally, in message 5, A encrypts the nonce from the previous message, and returns it to B.

The protocol for repeated authentication is as follows (we number the protocol messages $1'$, etc, to distinguish them from the messages of the initial protocol):

$$\begin{array}{lll}
\text{Message } 1'. & A \to B & : N_a', \{T_b, A, K_{ab}\}_{K_{bb}} \\
\text{Message } 2'. & B \to A & : N_b'', \{N_a'\}_{K_{ab}} \\
\text{Message } 3'. & A \to B & : \{N_b''\}_{K_{ab}}.
\end{array}$$

In message $1'$, when A wants to re-establish a session with B, he sends the ticket previously established, along with a nonce challenge N_a'. If the generalized timestamp is still valid, B encrypts N_a' and returns it in message $2'$, along with his own nonce challenge N_b''. Finally, A encrypts N_b'' and returns it in message $3'$.

One serious weakness with the repeated authentication protocol is that it will act as an encrypting oracle. Suppose the intruder has overheard a ticket from the initial protocol, and wants to encrypt some message M with the key. Then he can carry out the following exchange:

$$\begin{array}{lrcl} \text{Message } 1'. & I_A \rightarrow B & : & M, \{T_b, A, K_{ab}\}_{K_{bb}} \\ \text{Message } 2'. & B \rightarrow I_A & : & N_b, \{M\}_{K_{ab}}. \end{array}$$

Later, when A and B really do re-establish a session, the intruder may send the message $\{M\}_{K_{ab}}$ to A, who will understand it as having been sent by B (or vice versa). Note that if the message M is longer than the length of a nonce, then the intruder can repeat the above process several times to get B to encrypt each block; Clark and Jacob [9] have shown that a similar process works even if Cipher Block Chaining is used (see, for example, [27]).

Hwang et al. [14] present two attacks upon the Neuman-Stubblebine repeated authentication protocol [23]. The re-authentication part of the Neuman-Stubblebine protocol is identical to that of KSL, so one of their attacks also holds here:

$$\begin{array}{lrcl} \text{Message } \alpha.1'. & I_A \rightarrow B & : & N_i, \{T_b, A, K_{ab}\}_{K_{bb}} \\ \text{Message } \alpha.2'. & B \rightarrow I_A & : & N_b'', \{N_i\}_{K_{ab}} \\ \text{Message } \beta.1'. & I_A \rightarrow B & : & N_b'', \{T_b, A, K_{ab}\}_{K_{bb}} \\ \text{Message } \beta.2'. & B \rightarrow I_A & : & N_b''', \{N_b''\}_{K_{ab}} \\ \text{Message } \alpha.3'. & I_A \rightarrow B & : & \{N_b''\}_{K_{ab}}. \end{array}$$

In run α, the intruder imitates A to attack B. He meets the nonce challenge with N_b'' by using B as an oracle in run β.

One way to prevent the above attack is to implement the protocol so that B checks for replays of nonces it issued in current runs; this would prevent the above attack, because B would refuse to accept the nonce N_b'' in message $\beta.1'$. The above change seems to prevent the attack on the Neumann-Stubblebine protocol, but does not work for the KSL protocol.

It would seem that the protocol designers intended there to be some asymmetry between the two agents in the protocol: for each key, only one of the two agents is supposed to issue a ticket containing that key. However, as we show below, this is not the case.

$$
\begin{array}{llcl}
\text{Message } \alpha.1. & I_A \rightarrow B & : & N_i, A \\
\text{Message } \alpha.2. & B \rightarrow I_S & : & N_i, A, N_b, B \\
\text{Message } \beta.1. & I_B \rightarrow A & : & N_b, B \\
\text{Message } \beta.2. & A \rightarrow S & : & N_b, B, N_a, A \\
\text{Message } \beta.3. & S \rightarrow A & : & \{N_a, B, K_{ab}\}_{K_{as}}, \{N_b, A, K_{ab}\}_{K_{bs}} \\
\text{Message } \beta.4. & A \rightarrow I_B & : & \{N_b, A, K_{ab}\}_{K_{bs}}, \{T_a, B, K_{ab}\}_{K_{aa}}, \\
 & & & N_a', \{N_b\}_{K_{ab}} \\
\text{Message } \alpha.3. & I_S \rightarrow B & : & \{N_b, A, K_{ab}\}_{K_{bs}}, \{N_a, B, K_{ab}\}_{K_{as}} \\
\text{Message } \alpha.4. & B \rightarrow I_A & : & \{N_a, B, K_{ab}\}_{K_{as}}, \{T_b, A, K_{ab}\}_{K_{bb}}, \\
 & & & N_b', \{N_i\}_{K_{ab}}.
\end{array}
$$

The attack begins by the intruder imitating A and starting a run of the protocol with B (message $\alpha.1$). B responds by choosing a nonce N_b, and trying to forward it to the server, but the intruder intercepts this message (message $\alpha.2$). The intruder then starts a second run of the protocol, this time with A, imitating B, and sending the nonce N_b from the previous message (message $\beta.1$). A sends this nonce and its own nonce to the server (message $\beta.2$). The server chooses a session key K_{ab}, and prepares two components, one each for A and B (message $\beta.3$). A then tries sending to B its component, the ticket for re-authentication, a nonce challenge (N_a'), and the original nonce encrypted with the key (message $\beta.4$), and the intruder intercepts the message. The intruder can now reverse the order of the two components of message $\beta.3$, and forward them to B (message $\alpha.3$); note that this message is of precisely the form that B is expecting. Therefore, B creates a ticket containing K_{ab}, and tries sending it to A along with a nonce challenge (N_b') and the other appropriate components, but the intruder intercepts it (message $\alpha.4$). At this point the intruder is in possession of a pair of tickets, $\{T_a, B, K_{ab}\}_{K_{aa}}$ and $\{T_b, A, K_{ab}\}_{K_{bb}}$ for A and B respectively.

The intruder may now use the re-authentication protocol as an oracle to complete the two runs α and β:

$$
\begin{array}{llcl}
\text{Message } \gamma.1'. & I_A \rightarrow B & : & N_a', \{T_b, A, K_{ab}\}_{K_{bb}} \\
\text{Message } \gamma.2'. & B \rightarrow I_A & : & N_b'', \{N_a'\}_{K_{ab}} \\
\text{Message } \beta.5. & I_B \rightarrow A & : & \{N_a'\}_{K_{ab}} \\
\text{Message } \delta.1'. & I_B \rightarrow A & : & N_b', \{T_a, B, K_{ab}\}_{K_{aa}} \\
\text{Message } \delta.2'. & A \rightarrow I_B & : & N_a'', \{N_b'\}_{K_{ab}} \\
\text{Message } \alpha.5. & I_A \rightarrow B & : & \{N_b'\}_{K_{ab}}.
\end{array}
$$

The intruder starts a run of the re-authentication protocol with B, imitating A, sending A's nonce challenge and the ticket obtained above (message $\gamma.1'$). B responds by encrypting the nonce and returning it (message $\gamma.2'$). The intruder can forward the encrypted component (message $\beta.5$) to complete run β, thus fooling A into thinking that it has been communicating with B. Similarly, in run δ, the intruder imitates B in a run of the re-authentication protocol with A in order to answer B's nonce challenge from run α.

The intruder can also use the tickets obtained above to attack the re-authentication protocol:

$$\begin{array}{lrcl}
\text{Message } \epsilon.1'. & I_A \to B & : & N_i', \{T_b, A, K_{ab}\}_{K_{bb}} \\
\text{Message } \epsilon.2'. & B \to I_A & : & N_b'', \{N_i'\}_{K_{ab}} \\
\text{Message } \zeta.1'. & I_B \to A & : & N_b'', \{T_a, B, K_{ab}\}_{K_{aa}} \\
\text{Message } \zeta.2'. & A \to I_B & : & N_a'', \{N_b''\}_{K_{ab}} \\
\text{Message } \epsilon.3'. & I_A \to B & : & \{N_b''\}_{K_{ab}}.
\end{array}$$

The intruder can imitate A in a run of the re-authentication protocol with B, sending the ticket obtained above (message $\epsilon.1'$). B responds by sending a nonce challenge N_b'' (message $\epsilon.2'$). The intruder can then use A as an oracle by starting another run of the re-authentication protocol, sending the ticket and B's nonce challenge (message $\zeta.1'$). A responds by encrypting the nonce and returning it (message $\zeta.2'$). Finally, the intruder can forward this encrypted component (message $\epsilon.3'$) to achieve an error of authentication.

To prevent these attacks, it is prudent to make three changes: to change message 3 so that the two encrypted components have different forms; to make the encrypted components in messages 4, $2'$ and $3'$ different; and to include either A's or B's identity in these last three components.

7 Conclusions

In this paper we have presented attacks upon four security protocols. The vulnerabilities exploited in the attacks and the forms the attacks took are not particularly novel, and have appeared in previous attacks. We must ask, then, why insecure protocols are still appearing. One reason seems to be that protocol developers are failing to learn from the mistakes of others, and to spot when their own designs contain the same vulnerabilities. Another reason is that extant techniques for protocol analysis are insufficient, or are being incorrectly applied, or are not being applied at all.

The weaknesses in the protocols that were exploited by the attacks in this paper are of two types.

In the Woo and Lam protocol (Section 3), the intruder was able to replay part of a message 4 and have it interpreted as part of a message 5. Similarly, in the KSL protocol (Section 6), the intruder was able to replay the second encrypted component of a message 3 and have it interpreted as the first encrypted component of a message 3, and also replayed parts of message $2'$s and have them interpreted as a message 5 and a message $3'$.

Similar flaws have been the cause of attacks on other protocols, for example: the attacks on the adapted Yahalom protocol [7] in [31]; an attack on the Neuman-Stubblebine protocol [23], reported in [30, 14]; the attack on the Wide Mouthed Frog Protocol [7] in [2]; and a protocol due to Snekkenes [28].

In [1], Abadi and Needham present eleven principles for designing cryptographic protocols; the weakness discussed above represent a violation of principle 10:

> If an encoding is used to present the meaning of a message, then it should be possible to tell which encoding is being used. In the common case where the encoding is protocol dependent, it should be possible to deduce that the message belongs to this protocol, and in fact to a particular run of the protocol, and to know its number in the protocol.

We would add to this that when a message has more than one component, it should be possible to tell which component is which.

To prevent these attacks, we need to make sure that the different encrypted components in the protocols have distinct forms. Following [1], one way is to include in each encrypted component text such as: "This is the second encrypted component of message 4 of the Woo and Lam mutual authentication protocol", or possibly some more concise representation thereof.

In the Andrew protocol (Section 4), the weakness that was exploited was that it was not possible to tell whether a particular message 2 encrypted with the key K_{ab} was from A to B, or vice versa. Similarly, in the station-to-station protocol (Section 5), the weakness is that it is not possible to tell who is the intended recipient of the encrypted components.

Similar flaws have been the cause of numerous attacks on other protocols, for example: the attack on the CCITT X.509 in [7]; the attack on the Denning-Sacco protocol [10] reported in [1, 2]; two attacks on the SPLICE protocol [33] reported in [13] and [8]; and the attack on the Needham-Schroeder Public Key Protocol [22, 7] in [18, 19]. This is also the cause of the original attack on the Neumann-Stubblebine protocol [23] presented in [14], which is also an attack on KSL.

These weaknesses represent violations of principle 3 of [1]:

> If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.

(We prefer the term "encrypted component" to the term "message", to stress that the principal's identity should be included in each component whose meaning depends upon that identity.)

One proviso to the above principle concerns protocols that are designed to be resistant to an intruder guessing a weak key, such as a password (see, for example, [4, 17]). If messages encrypted with such a weak key include an agent's identity, then if an intruder correctly guesses the key, then he may easily verify his guess. (I am grateful to Paul Syverson for pointing this out.)

Given the number of protocols that have been suggested, only to later be found to be flawed, it is clear that we need to develop better ways of analyzing protocols.

The most well known approach is the use of BAN Logic [7]. This is good at catching certain sorts of protocol errors, particularly concerning freshness, and many protocol errors have been detected this way. Unfortunately, the use of the logic is prone to errors. As described above, the logic cannot catch errors caused by an intruder reordering components. Further, errors often arise in the idealisation stage, particularly where a message is idealised into a formula containing information not present in the message itself. Finally, it does not concern itself with matters of confidentiality. We would recommend the use of BAN Logic as a first stage of protocol analysis, but recognize that it does not detect all errors.

A few years ago, most new attacks on protocols were caused by a lack of freshness indicators. Thanks to BAN Logic, protocol designers now seem to appreciate this point, and are avoiding these mistakes. It is our experience that nowadays most new attacks are caused by violations of one of the principles set out in [1], particularly the two quoted above. We would recommend that paper as essential reading for any protocol designer—or protocol breaker.

Finally, various methods have been used to represent protocols as state machines, and then use a tool to test whether an insecure state can ever be reached; for example, the Interrogator [20, 16], the NRL Protocol Analyzer [16], or the use of the process algebra CSP and its model checker FDR [24, 19]. These approaches tend to be good at discovering attacks of the sort presented in this paper, caused by a lack of explicitness in the protocol messages. All the tools suffer from the fact that the state space they need to explore can be very large. Further, if a tool verifies that a particular system running the protocol is secure, this doesn't necessary imply that the protocol is secure when run on larger systems. However, these methods certainly help to detect errors, and can improve our confidence in the security of a protocol.

It is my hope that the attacks described in this paper will encourage protocol designers to learn from the mistakes of others, and to avoid introducing the same vulnerabilities into new protocols.

References

[1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. Research Report 125, Digital Equipment Corporation Systems Research Center, 1994.

[2] R. Anderson and R. Needham. Programming satan's computer. In J. van Leeuwen, editor, *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[3] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. *ACM Computer Communications Review*, 20(5):119–132, 1990.

[4] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Computer Security Conference on Research in Security and Privacy*, pages 72–84, 1992.

[5] R. Bird, I. Gopal, A. Herzberg, P. A. Janson, S. Kutten, R. Mulva, and M. Yung. Systematic design of two-party authentication protocols. In *Proceedings of Crypto '91: Advances in Cryptology*, volume 576 of *Lecture Notes in Computer Science*, pages 44–61. Springer-Verlag, 1991.

[6] R. Bird, I. Gopal, A. Herzberg, P. A. Janson, S. Kutten, R. Mulva, and M. Yung. Systematic design of a family of attack-resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679–693, 1993.

[7] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, 1989.

[8] J. Clark and J. Jacob. On the security of recent protocols. *Information Processing Letters*, 56(3):151–155, 1995.

[9] J. Clark and J. Jacob. Non-repeatability is not enough. In preparation, 1996.

[10] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[11] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[12] D. Gollmann. What do we mean by entity authentication? In *IEEE Symposium on Research in Security and Privacy*, 1996.

[13] T. Hwang and Y.-H. Chen. On the security of SPLICE/AS—the authentication system in WIDE Internet. *Information Processing Letters*, 53:97–101, 1995.

[14] T. Hwang, N.-Y. Lee, C.-M. Li, M.-Y. Ko, and Y.-H. Chen. Two attacks on Neuman-Stubblebine authentication protocols. *Information Processing Letters*, 53:103–107, 1995.

[15] A. Kehne, J. Schönwälder, and H. Landendörfer. A nonce-based protocol for multiple authentications. *Operating Systems Review*, 26(4):84–89, 1992.

[16] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2), 1994.

[17] T. M. A. Lomas, L. Gong, J. H. Saltzer, and R. M. Needham. Reducing risks from poorly chosen keys. *Operating Systems Review*, 23(5):14–18, 1989. Proceedings of the Twelfth ACM Symposium on Operating Systems Principles.

[18] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[19] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996.

[20] J. K. Millen, S. C. Clark, and S. B. Freedman. The interrogator: Protocol security analysis. *IEEE Transactions on software Engineering*, 13(2), 1987.

[21] S. P. Miller, C. Neumann, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. Project Athena Technical Plan Section E.2.1, MIT, 1987. Available from URL `ftp://athena-dist.mit.edu/pub/kerberos/doc/techplan.PS`.

[22] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[23] B. C. Neuman and S. G. Stubblebine. A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10–14, 1993.

[24] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, 1995.

[25] A. W. Roscoe. Intensional specification of security protocols. In *9th IEEE Computer Security Foundations Workshop*, pages 28–38, 1996.

[26] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 7(3):247–280, 1989.

[27] B. Schneier. *Applied Cryptography.* Wiley, 1994.

[28] E. Snekkenes. Roles in cryptographic protocols. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 105–119, 1992.

[29] J. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proc. Winter USENIX Conference, Dallas*, 1988.

[30] P. Syverson. On key distribution protocols for repeated authentication. *Operating Systems Review*, 27(4):24–30, 1993.

[31] P. Syverson. A taxonomy of replay attacks. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 131–136, 1994.

[32] T. Y. C. Woo and S. S. Lam. A lesson on authenticated protocol design. *Operating Systems Review*, 28(3):24–37, 1994.

[33] S. Yamaguchi, K. Okayama, and H. Miyahara. Design and implementation of an authentication system in WIDE Internet environment. In *Proc. 10th IEEE Region Conf. on Computer and Communication Systems*, 1990.